第 38 卷第 3 期
录用定稿

计算机应用研究
Application Research of Computers

Vol. 38 No. 3
Accepted Paper



# 高效用模式挖掘关键技术综述 *

张春砚，韩　萌†，孙　蕊，杜诗语，申明尧

(北方民族大学 计算机科学与工程学院，银川 750021)

**摘　要：**高效用模式挖掘(high utility pattern mining，HUPM)是近年来研究的新兴主题。效用的概念为分析人员挖掘相关项集提供了更大的灵活性，以用户的需求为出发点，从权重、值、数量和其他信息进行度量。通过分析有关 HUPM 最先进的方法，对其进行全面和结构化的概述。首先，通过介绍 HUPM 的相关概念、公式，并给出应用示例，对 HUPM 有更深一步的理解。针对用于挖掘不同类型 HUPM 的最常见和最先进的关键技术的进行分类，包括基于 Apriori，基于树，基于列表，基于映射，基于垂直/水平数据格式，基于索引等方法。针对现有关键技术的用途和优缺点，进行了全面概述。然后，由于静态数据难以满足实际需要，总结了在数据流上应用的 HUPM 方法，主要包括基于增量方法，基于滑动窗口模型方法，基于时间衰减模型方法，基于地标模型方法等。最后，给出了现在技术的不足和改进方向，并且有针对性的提出了新的研究方法。

**关键词：**综述；模式挖掘；高效用模式挖掘；数据流；增量数据

**中图分类号：**TP　　doi: 10.19734/j.issn.1001-3695.2020.02.0027

## Survey of key technologies for high utility patterns mining

Zhang Chunyan, Han Meng†, Sun Rui, Du Shiyu, Shen Mingyao

(*School of Computer Science & Engineering*, *North Minzu University*, *Yinchuan* 750021, *China*)

**Abstract:** High utility pattern mining (HUPM) is an emerging topic in recent years. The concept of utility provides greater flexibility for analysts to mine related itemsets, taking the user's needs as a starting point, measuring weights, values, quantities, and other information. This article provides a comprehensive and structured survey of the most advanced methods of HUPM by analyzing them. First of all, by introducing the relevant concepts and formulas of HUPM and giving application examples, this paper has a deeper understanding of HUPM. Classify the most common and advanced key technologies used to mine different types of HUPM, including Apriori-based, tree-based, list-based, projection-based, vertical/horizontal data-based, index-based, and more. This paper provides a comprehensive survey of the uses, advantages and disadvantages of existing key technologies. Then, because the static data is difficult to meet the actual needs, this paper summarizes the HUPM methods applied on the data stream, mainly based on the incremental methods, based on the sliding window model methods, based on the time decay model methods, based on the landmark model methods. Finally, this paper gives the shortcomings of the current technologies and the direction of improvement, and proposes new research methods in a targeted manner.

**Key words:** survey; pattern mining; high utility pattern mining; data stream; incremental databases

## 0　引言

模式挖掘是数据挖掘中的核心任务之一。频繁模式挖掘(frequent itemset mining, FIM)识别经常出现在事务数据集中的项集，并假设所有项都具有相同的重要性(单位利润，价格等)。但是，一个项在事务中只能出现一次或零次。传统的 FIM 将丢弃此信息，可能会挖掘许多低利润的频繁项集。

高效用项集挖掘(high utility itemset mining, HUIM)是 FIM 的一个重要领域。HUIM 考虑项和项集的数量和利润来衡量项目的“有用性”。如果数据库中项集的总效用不小于用户指定的最小效用阈值(minimum utility threshold, *minutil*)，则称为高效用项集(high utility itemsets, HUIs)。否则，它称为低效用项集。例如，在市场篮子分析的背景下，它包括找到产生至少等于某个最小效用价值的利润的所有项集。HUIM 的目标是识别对用户有意义的项或项集。因此，研究人员提出了许多挖掘 HUIs 的方法，以便迅速采取适当的措施。

在 HUIM 的初期，研究者提出了基于先验的高效用模式 TwoPhase[1]算法，这是经典算法之一。该算法提出了事务加权效用[1](transaction weighted utility, TWU)和事务加权向下闭包[1](transaction weighted downward closure, TWDC)的属性。在阶段 I 中，使用 TWDC 属性找到候选高事务加权效用项集(high transaction weighted utility itemset, HTWUIs)；在阶段 II 中，需要额外的数据库扫描来识别实际的 HTWUIs。该算法有效减少了候选项的数量，精确得到了高效用项集完备集，但是这种算法由于生成和测试方法导致执行时间和内存使用过多，并且需要大量的数据库扫描。为此，提出其他算法来克服 Two-Phase 算法的不足。作为其中之一，IHUP[3]算法使用 ihp-tree 数据结构，生成 HTWUIs，用于挖掘增量数据库[3]中的 HUIs。但是，它会产生大量候选项。随着技术不断的进步，HUP-growth[4]算法挖掘没有候选项生成的 HUIs，采用两阶段模型和 HUP-tree 结构来维护 1-HTWUIs，从而加快挖掘过程。PRE-HUI-DEL[5]算法，用于事前删除的预大概念(pre-large concept)更新高效用项集，从而加快更新信息的处理时间。PRE-HUI-INS[6]算法基于 pre-large 概念的特性，保留了

---

**收稿日期：**2020-02-13；**修回日期：**2020-03-27　**基金项目：**国家自然科学基金资助项目(61563001)；计算机应用技术自治区重点学科资助项目(PY1902)；宁夏高等学校一流学科建设(电子科学与技术学科)资助项目(NXYKXY2017A07)

**作者简介：**张春砚(1995-)，女，河北张家口人，硕士研究生，主要研究方向为数据挖掘；韩萌(1982-)，女(通信作者)，河南商丘人，副教授，硕导，博士，主要研究方向为数据挖掘(2003051@nmu.edu.cn)；孙蕊(1993-)，女，山东邹城人，硕士研究生，主要研究方向为数据挖掘；杜诗语(1996-)，女，辽宁抚顺人，硕士研究生，主要研究方向为数据挖掘；申明尧(1994-)，男，山东菏泽人，硕士研究生，主要研究方向为数据挖掘.

HTWUIs，以避免数据库重做，直到插入的事务的累积总效用达到安全界限。PIHUP[7]算法通过一个附加的阈值来处理动态添加的事务，对数据库只需要一次扫描，更适合处理动态数据。HUPPP[8]算法，将挖掘 HUPM 问题扩展到挖掘高效用部分周期模式，不仅考虑事件的发生时间顺序和周期长度，而且考虑事件的数量和个人利润。该算法使用两阶段周期效用上限模型来避免挖掘过程中的信息丢失。

因为两阶段算法在阶段 I 生成了大量的候选项，而且在阶段 II 引起了可扩展性问题，因此效率较低。虽然已经做了很多努力以减少阶段 I 产生的候选项数量，但当原始数据包含许多长事务或 *minutil* 很低时，挑战仍然存在。因此，单阶段算法的提出有效的缓解了这一问题。

HUI-Miner[9]算法是第一个发现 HUIs 的单阶段算法。它提出了一种垂直的数据结构效用列表[9]和剩余效用列表[9]的概念，并在许多新的 HUPM 算法中得到了广泛的推广。FHM[10]算法，考虑 2-项集之间的共现性，增强了 HUI-Miner 的剪枝特性。CHUI-Mine[11]算法是第一个在不产生候选项的情况下，从数据库中找到闭合高效用项集完整集合的紧凑算法。EFIM[12]算法依赖于两个新的上界：修订子树效用和局部效用，以更有效地修剪搜索空间。mHUIMiner[13]算法利用树结构来指导挖掘项集展开过程，以避免考虑数据库中不存在的项集，并且没有复杂的剪枝策略。MHUI[14]算法用来挖掘具有多个 *minutil* 的 HUIs。此算法引入了后缀最小效用的概念，并提出了有效挖掘 HUIs 的广义修剪策略。kHMC[15]算法发现 Top-k 高效用项集，采用真实项效用，共存效用降序和覆盖概念三种策略来有效地提高其内部 *minutil*，减小搜索空间。FOSHU[16]算法考虑物品的现货时间段，以及具有正/负项的物品。该算法使用效用列表和深度优先搜索，同时在所有时段挖掘 HUIs。HUPNU[17]算法在不确定数据库中，依赖于具有正项和负项的概率-效用列表来直接挖掘 HUIs 而无须生成和测试候选项。

近年来，用于挖掘 HUIM 的关键技术得到了广泛的研究。UDHUP-Apriori[18]算法以一种水平的方式挖掘最新的高效用模式，使用类似于 Apriori 的方法递归地派生 HUIs，从而加快了挖掘过程。动态环境中的事务插入增量 HUI-list-INS[19]算法，在不生成候选列表的情况下减少计算量，还采用了枚举树和 2-项集，加快了计算速度。PHUS[20]算法提出了最大效用度量，以简化一组序列中子序列的效用评估，并采用了有效的序列效用上限模型，以避免挖掘中丢失信息。该算法还设计了一种有效的基于映射的修剪策略，以产生更准确的子序列。CHUM[21]算法采用广义效用列表(generalized utility-list, gutility-list)，每个项集都包含这种结构,该效用列表结构是数据库垂直表示，能够计算项集的闭包并有效地生成候选项。

本文的主要贡献如下：

a)本文是对高效用关键技术的全面综述，并以系统的方式对其进行分析、总结；

b)本文深入，全面地总结了该领域的发展，介绍了 HUPM 的概念、公式以及对相关概念进行了比较说明，给出了应用示例。

c)因为文献[22]提到了 HUPM 的关键技术，所以本文是对关键技术的更深入的了解和介绍。并从不同的算法中，以不同的角度详细介绍了基于 Apriori，基于树，基于列表，基于映射，基于垂直/水平数据和基于索引的方法。

d)本文简要分析了数据流和增量数据库中高效用模式的方法。主要从三个方面进行描述，即基于滑动窗口模型的方法，基于时间衰减模型的方法和基于地标窗口模型的方法。

## 1　基本概念

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是一组项，$D=\{T_1,T_2,\cdots,T_n\}$ 是事务数据库，其中每个事务 $T_i$ 中的项是 $I$ 的子集。项集 $X$ 是 $I$ 的子集。事务 $T_q$ 中的项 $i_p$ 的数量由 $n(i_p,T_q)$ 表示。外部效用 $s(i_p)$ 是效用表中项 $i_p$ 的单位值(例如，利润)。事务 $T_q$ 中项 $i_p$ 的效用，由 $u(i_p,T_q)$ 表示，定义为 $n(i_p,T_q)s(i_p)$ 。令项集 $X$ 为 $I$ 的子集。事务 $T_q$ 中 $X$ 的效用，由 $u(X,T_q)$ 表示，定义为

$$u(X,T_q)=\sum\nolimits_{i_p\in X}u(i_p,T_q) \tag{1}$$

项集 $X$ 在数据库中的效用，由 $u(X)$ 表示，定义为

$$u(X)=\sum\nolimits_{T_q\in D\wedge X\subseteq T_q}u(X,T_q) \tag{2}$$

如果项集的效用不低于用户指定的 *minutil*，则该项集称为高效用项集。否则，它被称为低效用项集。HUIM 的任务就是找到所有高效用项集。由于效用不具有反单调属性，因此运用了事务效用[1](transaction utility, TU)和 TWU[1]的概念来修剪搜索空间。事务的事务效用，表示为 $tu(T_q)$ ，是 $T_q$ 中所有项的效用的总和：

$$tu(T_q)=\sum\nolimits_{i_p\in T_q}u(i_p,T_q) \tag{3}$$

项集 $X$ 的事务加权效用，表示为 $TWU(X)$是包含 $X$ 的所有事务的事务效用的总和：

$$TWU(X)=\sum\nolimits_{T_q\in D\wedge X\subseteq T_q}tu(T_q) \tag{4}$$

假设有一个小型事务数据库，如表 1 所示。表 2 显示了每个项的利润(外部效用)。表 1 中每行的值表示在特定事务中购买的每个项的数量(即局部事务效用)。最后一列显示了每个事务的事务效用。

在事务 $t_1$ 中，购买了 2 个 $A$，1 个 $C$ 和 1 个 $D$，产生了 80 元的事务效用，项 $A$ 的效用 $u(A,t_1)=2\times10=20$ 元。在整个数据库中项 $A$ 的效用 $u(A)=2\times10+2\times10+1\times10=50$ 元。在事务 $t_1$，$t_2$ 和 $t_5$ 中，项集 $AC$ 发生了 3 次。在事务 $t_1$ 中，$u(AC,t_1)=2\times10+25=45$ 元；在事务 $t_2$ 中，$u(AC,t_2)=2\times10+25=45$ 元；在事务 $t_5$ 中 $u(AC,t_5)=1\times10+25=35$ 元；在整个数据库中项集 $AC$ 的效用 $u(AC)=45+45+35=125$ 元。假设 *minutil* 为 80，$u(BC)=1\times30+1\times25=55$，$u(CD)=1\times25+1\times35+1\times25+1\times35=120$。因此，项集 $CD$ 是 HUIs，而项集 $BC$ 不是 HUIs。项 $B$ 的 $TWU$ 是事务 $t_2$ 和 $t_4$ 效用的总和，$TWU(B)=tu(t_2)+tu(t_4)=75+30=105$ 元，项集 $CD$ 的 $TWU$ 是事务 $t_1$ 和 $t_3$ 效用的总和，$TWU(CD)=tu(t_1)+tu(t_3)=80+60=140$ 元。

表 1　事务数据库

Tab. 1　Transaction database

| TID | A | B | C | D | Transaction utility(yuan) |
|---|---|---|---|---|---|
| $t_1$ | 2 | 0 | 1 | 1 | 80 |
| $t_2$ | 2 | 1 | 1 | 0 | 75 |
| $t_3$ | 0 | 0 | 1 | 1 | 60 |
| $t_4$ | 0 | 1 | 0 | 0 | 30 |
| $t_5$ | 1 | 0 | 1 | 0 | 35 |

表 2　效用表

Tab. 2　Utility table

| Item | Profit(yuan) |
|---|---|
| A | 10 |
| B | 30 |
| C | 25 |
| D | 35 |

## 2　高效用模式挖掘关键技术

近几年，研究者们已经提出了大量的算法从数据中挖掘 HUPM。本章选择一些经典的，有代表性的、最新的 HUPM 算法，根据不同的挖掘原则和数据结构，对算法用到的关键技术进行分类。具体而言，为了便于讨论，本章将这些工作分为以下几类：

a) 基于先验(Apriori)的方法；

b) 基于树(tree)的方法；

c) 基于映射(projection)的方法；

d) 基于列表(list)的方法；

e) 基于垂直(vertical)/水平(horizontal)数据格式和索引(index)的方法。

## 2.1 基于先验(Apriori)的方法

研究人员提出了一个众所周知的向下闭合属性，也称为Apriori[23]属性，该属性指定频繁项集的所有非空子集都必须是频繁的，而一个不频繁项集的任何超集都不是频繁的。

为了将 Apriori 的向下闭合性质应用于效用问题，研究者们设计了 Two-Phase[1]算法，并引入了 TWDC 和 TWU 的两个属性来发现 HUIs。在第一阶段，该算法使用候选项先生成后测试策略来查找所有 TWU 不小于 *minutil* 的项集。然后，在第二阶段，该算法扫描数据库以查找在第一阶段找到的项集的实际效用值。TWU 属性不仅限制了搜索空间，还涵盖了所有 HUIs。

随着相关研究人员的不断钻研，探索了基于 Apriori 方法的其他高效用模式，例如高效用平均模式和高效用闭合模式。FUP[24]算法采用向下闭合属性，逐级搜索高效用平均项集。利用这一属性，在每个级别生成的候选项集的数量大大减少，搜索空间也会减小。PHUI-UP[25]算法基于类似 Apriori 的方法和设计的上限模型挖掘潜在高效用项集(potential high-utility itemsets, PHUIs)。在不确定数据库中，该算法可以在未来的工作中被研究人员用作最先进的算法之一。随着 HUPM 研究的不断发展，基于 Apriori 方法也被用于挖掘闭合项集。CHUD[26]算法基于 Two-Phase[1]算法，该算法在第一阶段提取可能的高效闭合项集的集合，并在第二阶段计算该集合的实际效用信息。

总之，所有早期的 HUPM 方法都改进了基于 Apriori 的算法。Apriori 使用逐步的候选项生成和测试方法。优点是使用 Apriori 方法可以删除大量候选对象，提高挖掘有用模式的效率，并且在恢复所有 HUI 方面也具有良好的性能，例如算法 Two-Phase[1]和 CHUD[26]；还可以减少重新处理整个更新数据库的时间，例如算法 FUP[16]。但是，基于 Apriori 的算法也有很多缺点，例如多次数据库扫描，例如算法 Two-Phase[1]和 PHUI-UP[25]。由于在阶段 I 中生成了大量候选集，因此消耗了大量内存，例如算法 CHUD[26]和 FUP[24]。具体算法如表 3 所示。

表 3　基于 Apriori 的 HUPM 算法

Tab. 3　Apriori-based HUPM algorithm

| 算法名称 | HUIs 类型 | 数据集 | 优点 | 缺点 |
|---|---|---|---|---|
| Two-Phase[1](2005) | 全集 | T20I6D1000K, Chain-store | 删除了大量候选项。 | 测试后将搜索候选级别，并且需要多次数据库扫描。 |
| FUP[24](2009) | 平均项集 | Chain-store | 减少重新处理整个更新数据库的时间。 | 生成大量候选项。 |
| CHUD[26](2011) | 闭合项集 | Mushroom, Foodmart, BMSWebView1, T10I8D200K | 它可以减少大量不需要的 HUIs。此外，当可以恢复所有 HUIs 时，本文中提到的组合优于当前的最新算法。 | 该算法消耗大量内存，并且在事务交叉点中消耗更多的运行时间。当存在大量候选高效用项集时，算法会降低性能。 |
| PHUI-UP[25](2016) | 潜在项集 | T10I4D100K, Foodmart, Accident, Retail | 对于不确定的数据库，这是一个改进。 | 存在重复扫描数据库的问题。 |

## 2.2 基于树(tree)的方法

尽管 Apriori 方法可以有效地挖掘 HUIs，但是存在诸如生成大量候选项，反复扫描数据库以及运行缓慢等问题。为了避免这些缺点，提出了基于树的 HUIM 算法。这些基于树的算法包括三个步骤：1)构建树；2)使用算法从树生成候选 HUIs；3)从候选集中识别 HUIs。

基于树的方法广泛用于静态数据库中。UP-Growth[27]和 UP-Growth+[28]可以通过一种改进的高估方法来减少要提取的候选项的数量，它们需要两次数据库扫描来建立自己的树结构，即 UP-Tree。UFCP-Miner[81]算法是根据 UP-Growth 算法改进的，将频率因素和效用因素同时考虑进高效用模式挖掘中。通过两次扫描数据集计算效用值，并将事务项集整理到 UFCP-Tree 上。CTU-PRO[29]算法通过从底部到顶部遍历压缩的效用模式(compact utility pattern, cup)树来挖掘 HUIs。TWU 概念用于修剪 CTU-PRO 中的搜索空间，但是避免了重新扫描数据库以确定 HTWUIs 的实际效用。该算法通过构建可在磁盘上独立挖掘的并行细分来适应较大的数据集。近来，已经开发了许多新颖的树结构以改善挖掘的 HUPM 的性能。USpan[30] 算法使用序列加权效用(sequence-weighted utilization, SWU)和序列加权向下闭合属性(sequence-weighted downward closure property, SDCP)构造了词典顺序序列树(lexicographic quantitative sequence tree, LQS-tree)结构，然后提取完整的 HUSP(high utility sequence pattern)。HIMU[31]算法使用多个 *minutils* 来挖掘 HUIs，提出了多个项效用集合枚举树(multiple item utility Set-enumeration, MIU-tree)以及 MIU-tree 中 HUIs 的全局和条件向下闭合(global downward closure, GDC 和 conditional downward closure, CDC)属性。该算法比使用单个 *minutil* 更灵活，更现实。dHAUIM[32]算法使用一种称为 IDUL 前缀树的新结构，通过递归过程快速计算项集的平均效用和效用上限，以维持较高的效用平均项集。基于高效用平均模式，研究人员提出了 MAU-Growth[33]算法，该方法应用 MAUTree 从数据库中挖掘高效用平均的稀有模式。该算法考虑了模式的长度，以便有效地减少模式对其自身长度的依赖性，以便挖掘比以前算法挖掘的模式更有意义的稀有模式。REPT[34]算法通过挖掘前 $k$ 个 HUIs，大大减少了候选项的数量。该算法采用三种策略构建全局树和增加最小阈值，有效地减少搜索空间。

随着大数据时代的到来，研究人员已使用基于树的方法来解决增量数据库和数据流中的问题。IHUP[3]算法是 HUPM 中用于解决增量数据库挖掘的最先进的算法之一。它使用单个通道构造自己的树结构，称为 IHUP-Tree，并根据传统的高估方法查找所有 HUIs。然而，由于该方法的应用，它产生了大量的候选项。MAHUSP[35]算法设计了一种有效的树结构，即 MAS-Tree，用于在数据流上存储潜在的 HUSP。该算法不仅可以有效地发现数据流上的 HUSP，而且在牺牲所发现的质量的前提下适应了内存分配。从研究人员的不断发现中，现有算法需要多次数据库扫描以挖掘 HUIs，这会降低其效率。HUM-UT[36]算法用于从事务数据流中查找 HUIs，并提出了一种新的数据结构 UT-Tree。该结构是通过数据库扫描创建的，并且效用信息仅存储在尾节点上，以维护事务中项集的效用信息，以避免进行多次数据库扫描。

在 MAHUSP[35]算法中，实验都用 Java 实现，并且在具有 16GB RAM 的 Intel(R)i7 2.80GHz 计算机上进行的。假设 *eHUSP* 是真实的高效用序列模式集，*appHUSP* 是返回的高

效用序列模式的近似集，*ApproxUtil*(*P*)被用作模式的近似效用，而 *TureUtil*(*P*)为模式的真正效用。使用以下性能指标来评估算法的有效性：

a) Precision：数据流的平均精度：

$$\text{precision}=\frac{|\text{appHUSPs}\cap\text{eHUSP}|}{\text{appHUSPs}} \tag{5}$$

b) *Recall*：数据流的平均召回值：

$$\text{recall}=\frac{|\text{appHUSPs}\cap\text{eHUSP}|}{\text{eHUSP}} \tag{6}$$

c) *F-Score*：精度和查全值的调和平均值：

$$F\text{-Score}:2\times\frac{\text{precision}\times\text{recall}}{\text{precision}+\text{recall}} \tag{7}$$

d) *AvgTrueUtil*：该方法返回的模式的平均实际效用。

e) *Relative Utility Error*：是通过与方法返回的模式的确切效用进行比较而得出的平均效用错误：

$$\text{relative utility error}=\frac{\sum_{P\in\text{appHUSPs}}\dfrac{\text{ApproxUtil}(P)-\text{TrueUtil}(P)}{\text{TrueUtil}(P)}}{|\text{appHUSPs}|} \tag{8}$$

f) *AvgLength*：平均长度，该方法使用此度量标准，指示返回的模式的平均长度接近通过精确方法获得的模式的平均长度。

g) *Run Time*：方法在输入数据流上的总执行时间。

h) *Memory Usage*：该方法的内存使用情况。

总而言之，基于树的算法的优势在于，a)对于包含密集数据集和长模式稀疏数据的较大数据集的性能更好，例如算法 CTU-PRO[29]；b)可以有效减少候选对象的数量，避免重复扫描数据库，例如算法 UP-Growth[27]，UP-Growth+[28]，REPT[34]，HIMU[20]和 HUM-UT[36]等；c)树结构中使用的树节点数量很少，并且适合于内存分配，例如算法 IHUP[3]，MAHUSP[35]。尽管这些树结构通常很紧凑，但它们可能不是最小的，仍然会占用大量存储空间。这些方法的挖掘性能与整个挖掘过程中构造的条件树的数量以及构建/遍历每个条件树的成本密切相关。因此，这些算法的性能瓶颈之一是生成大量具有高时间和空间成本的条件树。缺点有：a)生成了大量候选项，例如算法 CTU-PRO[29]和 IHUP[3]；b)检查和存储最小节点效用需要花费时间和内存，例如算法 UP-Growth[27]；c)树结构的处理过程非常耗时，例如算法 REPT[34]，HIMU[20]，HUM-UT[36]和 MAHUSP[35]。具体算法如图 1 所示。

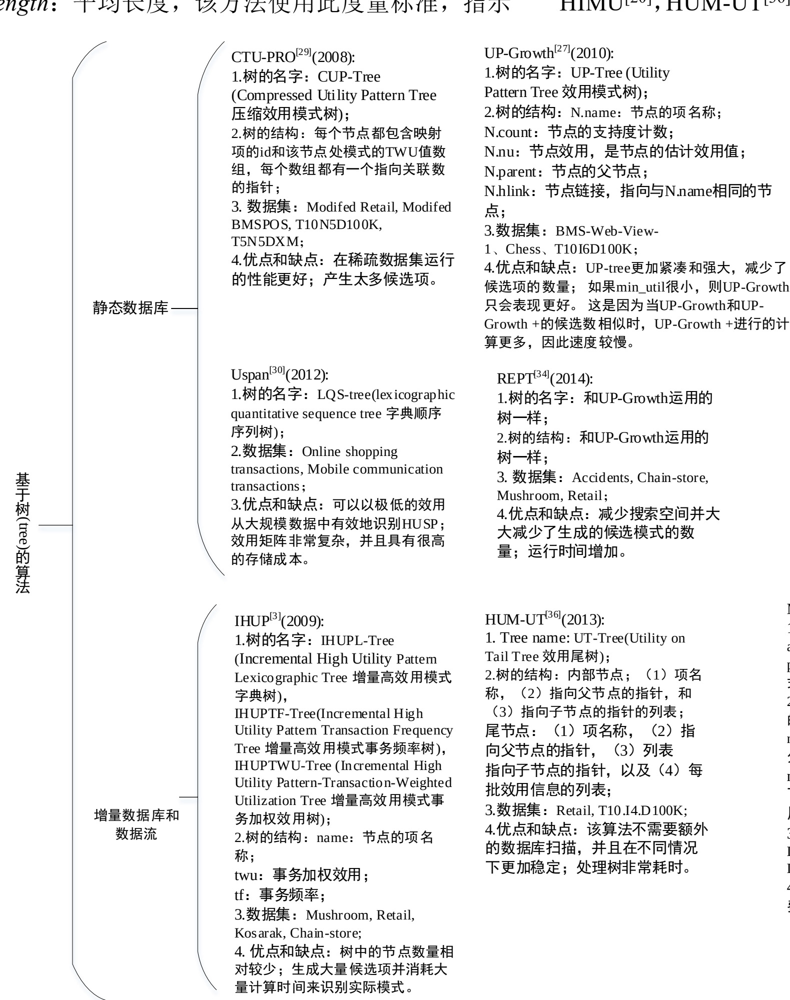



图 1 基于树(tree)的 HUPM 方法

Fig. 1 Tree -based HUPM method

## 2.3 基于映射(projection)的方法

为了克服基于树的方法的缺点，研究人员提出了一些基于映射的方法来提高挖掘性能，这些方法已广泛用于数据挖掘中。总体思路是将处理后的数据库递归投影到一些较小的映射子数据库中。然后在每个映射子数据库中，增长项集或子序列片段[22]。

当主内存不足以处理大型数据集时，研究人员将使用并行映射方案来使用磁盘存储。CTU-PROL[29]算法使用一个可以独立挖掘的并行投影为太大而无法保存在主存储器中的数据集创建细分。TWU 的反单调性用于减小 CTU-PROL 中细分的搜索空间。

基于前缀的投影方法，可以有效地提高效用上限，并可以优化挖掘过程。PHUS[20]算法扩展了 PrefixSpan[37]，并使用基于投影的修剪策略来实现序列效用的紧凑上限。并且提出了最大效用度量和序列效用上限(sequence utility upper bound, SUUB)模型的概念。因此，它可以避免考虑过多的候选项，

而可以使用 SUUB 模型来提高挖掘 HUSP 的性能。TKHUP_MaR[84]算法采用前缀投影结构，进行并行挖掘。即通过两次扫描数据库，利用三次 MapReduce 来实现并行 top-k 高效用模式的挖掘。

在实际研究中，研究人员不仅使用映射方法来挖掘有效的项集,还将它们与事务合并相结合以进一步提高挖掘性能。EFIM[12]算法是一种基于单阶段映射的高效用算法。为了降低数据库扫描的成本，EFIM 还提出了数据库投影和事务合并方法，即高数据库投影(high database projection, HDP)和高事务合并(high transaction merging, HTM)。为了处理动态单位利润数据库中的所有 HUIs，设计了 EFIM 的扩展算法 MEFIM[38]。它依靠数据库投影和另一种新颖的紧凑型数据库格式来有效地发现所需的项集。CHN[39]算法也应用了这种方法，并引用了基于子树的修剪策略[40]，该策略减少了修剪搜索空间并加快了挖掘过程。在研究较大的项集时，投影和合并将减小数据库的大小。EHNL[41]算法也将上述方法以及负面效用和长度限制应用于挖掘 HUIs。EHIN[42]算法使用数据集投影和合并技术来减少内存需求并加快挖掘过程的执行时间。在映射数据集之前和之后执行两次事务合并。这些技术减少了搜索空间。ETKHUP[82]算法在数据库投影过程中，应用事务排序及合并策略减少运行时间和内存消耗，从而高效得挖掘 Top-k 项集。此算法设定的 *minutil* 由用户指定高效用模式个数，而不是人为设定阈值。

总之，基于投的算法具有避免重新扫描数据库并减少扫描成本的优势，例如算法 CTU-PROL[29]，EFIM[12]，CHN[39]和 EHNL[41]，ETKHUP[82]。在基于投影的修剪策略中，可以获得更准确的子序列序列效用上限，因此，修剪效果和执行效率非常好，例如算法 PHUS[20]。缺点是会生成大量冗余候选项，例如算法 CTU-PROL[29]，PHUS[20]，CHN[39]和 EHNL[41]。具体算法如表 4 所示。

表 4　基于映射(projection)的 HUPM 方法

Tab. 4　Proiection-based HUPM algorithm

| 算法名字 | 特定方法 | 数据集 | 优点 | 缺点 |
|---|---|---|---|---|
| CTU-PROL[29](2008) | 并行映射 | Modifed Retail, Modifed BMSPOS, T10N5D100K, T5N5DXM | 避免重新扫描数据库以确定高事务加权项集的实际用途。 | 生成了大量候选项，并且算法使用的资源可能很高。 |
| PHUS[20](2014) | 基于前缀映射 | S8T6I4N4KD200K | 在修剪效果和执行效率上均表现出色。 | 生成大量候选项。 |
| TKHUP_MaR[84](2017) | 基于前缀映射 | Connect, Pumsb, Accidents, Chain_store | 该算法验证了在并行环境下挖掘指定数目的高效用模式的有效性及正确性。 | 在处理较小数据集时，挖掘效率不如单机模式下的效率。 |
| EFIM[12](2017) | 映射和事务合并 | Accident, BMS, Chess, Connect, Foodmart, Mushroom | 消耗的内存更少，并且复杂度与搜索空间中的项数大致成线性关系。 | 有时递归映射很耗时，并且会占用大量内存。 |
| CHN[39](2018) | 映射和事务合并 | Accidents, Chess, Mushroom, Pumsb, BMSPOS, Retail, kosarak, T40I10D100K, T10I4D100K | 在密集和稀疏数据集上表现良好。 | 使用 TWU 属性，会生成大量冗余候选项。 |
| EHNL[41](2019) | 映射和事务合并 | Accidents, Chess, Mushroom, T40I10D100K | 数据集映射和事务合并技术可降低扫描成本。 | 生成大量候选项，映射有时会浪费时间。 |
| ETKHUP[82](2019) | 映射和事务合并 | Chess, Mushroom, Connect, T25I10D10K, T20I6D100K, Foodmart | 减少了运行时间和和内存消耗，尤其适用于密集数据集。 | 在稀疏数据集上性能较差。 |

### 2.4　基于列表(list)的方法

在 HUPM 中，除了基于树的方法之外，研究人员还探索了基于列表(list)的方法。挖掘步骤如下：1)对数据库执行扫描，以为每个项集构建效用列表；2)再次扫描数据库，在效用列表中修改事务；3)删除小于 *minutil* 的项集，并减少搜索空间。基于列表的方法可以清楚地维护有关事务中项集的信息，并且可以快速计算项集的效用，并缩短搜索时间。

HUI-Miner[9]算法使用一种称为效用列表的新颖结构来存储有关项集的效用信息和用于修剪搜索空间的启发式信息。HUI-Miner 有效地从内置效用列表中挖掘 HUIs，从而避免了大量候选项集的昂贵生成和效用计算。根据表 5 和 6，在第二次数据库扫描期间，该算法为表 5 项集{*AB*}和表 6 项集{*BC*}构造效用列表。

表 5　项集{AB}的效用列表

Tab. 5　Utility list for itemset {AB}

| Tid | Iutil | Rutil |
|---|---|---|
| 1 | 30 | 40 |
| 4 | 50 | 40 |
| 5 | 30 | 30 |

表 6　项集{BC}的效用列表

Tab. 6　Utility list for itemset {BC}

| Tid | Iutil | Rutil |
|---|---|---|
| 3 | 50 | 40 |
| 5 | 50 | 0 |

项集 *X* 的效用列表中的每个元素都包含三个字段：*tid*，*iutil* 和 *rutil*[9]。*Tid* 代表包含 *X* 的事务 *T*；*iutil* 是 *X* 在 *T* 中的效用，即 *iutil*(*X*,*T*)；*rutil* 是 *X* 在 *T* 中的剩余效用，即 *rutil*(*X*, *T*)。

由于效用列表的引入，许多算法使用此结构来挖掘 HUIs，从而提高了挖掘性能。HUI-list-INS[19]算法继承了 HUI-Miner[9]算法，并构建了一个效用列表结构，用于在增量数据库中挖掘 HUIs，以维护和更新已发现的 HUIs 以及进行事务插入。HUI-list-DEL[43]算法是一种通过使用动态数据库中删除的记录的内置效用列表结构来发现 HUIs 的算法。在这种算法中，可以直接生成新的 HUIs，而无须生成候选对象和进行大量的数据库扫描。

随着 HUPM 的不断发展，效用列表已无法满足算法的需求。研究人员根据效用列表提出了许多扩展结构，以进一步提高性能。HUP-Miner[44]算法引入的分区效用列表数据结构，此结构借鉴了 tid-list[45]表示的基本思想。它也是效用列表的扩展。通过执行项集 *Rx* 和 *Ry* 的 tid-list 的交集来计算项集 *Rxy* 的分区效用列表。这个过程与 HUI-Miner[9]算法非常相似。该算法使用的 LA-Prune(基于正向修剪概念)策略为 $k$ 个项集提供了更严格的效用上限，因此可以修剪大量低效用项集，从而限制了挖掘 HUIs 的搜索空间。CHUI-Miner[11]算法使用扩展效用列表(extend utility list, EU-List)的新结构来维护事务中项集的效用信息，这使得原始数据库不会被扫描，并有效地计算了内存中的项集效用和效用单位数组。该算法使用分而治之的方法来挖掘数据库中完整的 CHUI 集，而不生成

候选对象。CHUM[21]算法提出了一个通用效用列表(generalized utility-list, gutility-list)，用于存储效用信息和有关搜索空间修剪的启发式信息，这与 HUI-Miner[9]算法中提出的效用列表不同，因为 gutility-list 可以快速计算闭合项集的效用。ULB-Miner[46]算法使用设计的效用列表缓冲区结构来有效存储和检索效用列表，并在挖掘过程中重新使用内存。线性时间方法还用于在效用列表缓冲区中构造效用列表段。LHUI-Miner[47]算法的目标是根据局部效用表(local utility list, LUlist)发现局部高效用项集(local high utility itemsets, LHUIs)。由于项集的效用随时间变化，因此希望找到一个时间点，在这些时间点，项集的效用会显着变化(增加或减少)。因此，扩展的 PHUI[47]算法挖掘高峰期的高效用项集。它包括一个时间段，在该时间段中，查找项集具有很高的效用。另外，由于 PHUIs(peak high utility itemsets)的集合可能很大，并且 PHUIs 中的某些项对其峰值的贡献不大，因此 NPHUI-Miner[47]算法用于挖掘一组非冗余的峰值高效用项集。MHAI[48]算法提出了一种新的列表结构，称为高效用平均项集列表(high average itemset list, HAI-List)。这种结构可以紧凑地捕获必要的信息，从而允许算法从给定的事务数据库生成 HAUIs，而无须生成候选集。HUOPM[49]算法在频率，效用和占用率方面考虑了用户偏好。该算法使用效用占用列表(utility-occupancy list, UO-list)和频率效用表(frequency-utility table, FU-table)来存储有关数据库的信息，以挖掘高效占用模式(high utility occupancy pattern, HUOP)。所提出的方法可以有效地发现完整的 HUIs 集，而无须生成候选对象。DMHUPS[50]算法利用 IUData-List 的数据结构，该结构存储 1-项集的信息及其在事务中的位置，以有效地获取初始数据库。另外，该算法同时计算多个有希望的候选项的效用，从而获得更严格的扩展上限，避免了生成冗余项，并找到多个有效模式。CRUSP[51]算法应用了删除效用列表(removed-utilities list, RUL)和删除效用位置列表(removed-utility-positions list, RUPL)。此列表指定了唯一需要考虑的项，它们可以作为所讨论的顺序模式串联的可能候选项，或在搜索树中显示为后代的任何顺序模式。MUHUI[52]算法基于概率效用列表(probabilistic utility list, PU-list)结构，该结构可以直接在不确定的数据库中挖掘 PHUIs，而无须生成候选对象，并且可以通过有效的修剪策略来避免为许多不需要的项集构建 PU-list，这大大提高了性能。FHN[53]算法依靠一种新颖的列表结构，即正负效用列表(positive and negative utility list, PNU-list)，同时考虑了正面和负面的单位利润，其目的是维护挖掘 HUIs 所需的所有信息，从而允许算法直接挖掘 HUIs 而不生成候选对象和无须执行多个耗时的数据库扫描。HUPNU[17]算法基于带有正负利润的概率-效用列表(positive and negative utility list, PU±-list)挖掘 HUIs 的正负单位利润。构造 PU±-list 时，可以修剪许多没有预设的早期项，以减少搜索空间。EHUSN[83]算法提出 1-2-UM 和 2-2UM 结构模型挖掘含负项的高效用模式，结合效用信息列表能快速剪枝非候选序列，从而是挖掘算法在时空效率上得到提升。IMHUP[54]算法使用索引效用列表(index utility list, IU-list)来使用新提议的项联接操作更有效地发现 HUIs，而无须进行任何比较。

总而言之，基于列表的算法的优点有：1)HUI-Miner[9]算法引入了剩余效用的概念和垂直数据结构的效用列表。随后，许多算法使用效用列表结构，例如 HUI-list-INS[19]和 HUI-list-DEL[43]，它们可以减少内存消耗并避免多次数据库扫描；2)扩展的效用列表结构和其他列表结构可以减少内存消耗和效用列表之间的连接操作，例如算法 ULB-Miner[46]，IMHUP[54]，HUP-Miner[44]，MUHUI[52]；3)一些新颖的算法在特定时间段内可用于挖掘传统 HUIM 无法找到的模式，从而减少运行时间和内存消耗，例如算法 LHUI-Miner[47]；4)并在一定程度上扩展了占用率以评估事务数据库，为效用挖掘提供了新的研究视角，例如算法 HUOPM[49]。因此，尽管大多数基于列表的方法可以加快挖掘速度，并且在稀疏和密集的数据库上具有良好的性能，但缺点是列表之间的连接需要高昂的成本，耗时的时间以及过多的内存使用等。例如，在 HUI-Miner[9]中，$(k+1)$项集的效用列表和 $k$ 项集的效用列表之间的连接非常耗时，导致运行时间较长。基于效用列表的扩展结构或其他列表结构，存在诸如复杂的构建过程的问题，具体有必须显示占用大量内存的数据集分区的数量；构造列表的过程更加复杂；动态调整参数具有挑战性，例如算法 HUP-Miner[44]，ULB-Miner[46]，LHUI-Miner[47]。具体算法如图 2 所示。

### 2.5 基于数据格式(data format)的方法

为了克服上述方法的缺点，研究人员最近提出了基于水平和垂直数据结构以及索引结构来挖掘 HUIs。一方面可以加快数据挖掘的进度，另一方面可以提高挖掘的性能。

水平数据结构是最基本的数据结构，运用在很多算法中。UtilityLevel[55]算法采用了水平方向的候选生成和测试机制挖掘 HUSP。因此，它生成大量候选序列并需要多次数据库扫描。SPHUITP[56]算法以水平方式挖掘短期高效用模式。这些模式定期出现，有利可图，并且在约束期间可以高效使用。

与 Eclat[57]算法挖掘频繁项集一样，HUI-Miner[9]算法提出了具有垂直数据结构的表结构。该算法首先通过构造效用列表来检查所有 1-扩展项集，然后在从扩展集合中识别并输出 HUIs 之后，逐个递归地处理有希望的扩展项集并放弃其他扩展项集。FHM[10]算法是 HUI-Miner[9]算法的一个增强版本，运用同样的垂直数据结构，加快挖掘进程。CHUM[21]算法采用数据库垂直表示，以便在不访问数据库的情况下加速生成项集闭包并计算其效用信息的执行时间，并有效地生成订单保留生成器。MHUI[14]算法用于有效挖掘具有多个 *minutil* 的 HUIs。所提出的算法利用垂直数据库表示来有效地存储项集信息，并且引入了后缀最小效用(suffix minimum utility, SMU)的新概念，以高效地挖掘 HUIs。

ISR-MOEA[58]算法，是一种基于索引集合表示的多目标进化算法，用于挖掘多样化的 Top-k HUIs。在算法中，建议使用索引集合个体表示方案来快速编码和解码 Top-k 模式集。IHUI-Mine[59]算法使用 subsume[60]索引来枚举所需的 HUIs 并修剪搜索空间。HUI-MMU[62]算法用于挖掘具有多个 *minutil* 的 HUIs。改进的 HUI-MMUTID[61]算法，基于 TID 索引策略，即 HUIMMUTID，以加快挖掘过程。

总而言之，基于数据格式的算法的优势在于，基于水平或垂直表示的算法可以大大减少找到的模式数量，有效地识别数据库中的 HUIs，并避免“稀有项目问题”，例如算法 SPHUTTP[56]，HUI-MMUTID[61]。基于索引的新颖算法可以实现多目标进化，并探索各种 top-k 高效用模式以进一步提高用户满意度，例如算法 ISR-MOEA[58]。缺点是需要多次扫描数据库以挖掘 HUIs，并且内存消耗很大，例如算法 UtilityLevel[55]，SPHUTTP[56]，CHUM[21]，HUI-MMUTID[61]；挖掘具有多个最小效用阈值的 HUIs 的算法可能对 *minutil* 的选择不敏感，例如算法 MHUI[14]。具体算法见表 7。

## 3 基于数据流的 HUPM 的方法

论文的前三章提出的大多数算法都适用于静态数据。随着物联网，云计算和大数据等技术的快速发展，数据流被广泛应用于网络，外贸管理和医学数据分析等众多领域。与静态数据相比，数据流具有一些独特的属性，例如到达速率快、不受限制以及无法回溯先前的事务。因此，对于不同的使用目的，数据流中有三种常用的模型：滑动窗口模型，时间衰减模型和地标模型[62, 63]。

效用列表

HUI-Mine[9](2012):
1.列表名字：Utility list(效用列表);
2.列表结构：tid：表示包含X的事务T；
iutil：是X在T中的效用，即u(X, T)；
rutil：是X在T中的剩余效用，即ru(X, T)；
3.数据集：Chain, Accidents, Chess, Kosarak, Mushroom, Retail, T10I4D100K , T40I10D100K；
4.优点和缺点：首次引入了剩余效用的概念和垂直数据结构的效用数据列表；(k + 1)-项集和k-项集的效用列表之间的连接非常耗时。

HUI-list-INS[19](2015):
1.列表名字：Utility list(效用列表);
2.列表结构：tid：表示包含X的事务T；
iutil：是X在T中的效用，即u(X, T)；
rutil：是X在T中的剩余效用，即ru(X, T)；
3.数据集：Foodmart, Retail, Chess,T10I4D100K；
4.优点和缺点：低内存消耗和许多生成的模式；运行时间长。

HUI-list-DEL[43](2016):
1.列表名字：Utility list(效用列表);
2.列表结构：tid：表示包含X的事务T；
iutil：是X在T中的效用，即u(X, T)；
rutil：是X在T中的剩余效用，即ru(X, T)；
3.数据集：Foodmart, Retail, Mushroom, T10I4D100K；
4.优点和缺点：可以避免多次数据库扫描，更适合实际应用；运行时间长。

扩展效用列表

HUP-Miner[44](2015):
1.列表名字：partition utility list(分区效用列表);
2.列表结构：Pk：分区；
UP(X, Pk)：分区中项集的效用；
RUP(X, Pk)：分区中项集的剩余效用；
3.数据集：Chain, Kosarak, Retail, Mushroom, Chess, T10I4D100K, T20I6D100K, T40I10D100K；
4.优点和缺点：这两种新的修剪策略可以减少效用列表之间的连接； 需要显式设置占用大量内存的数据集分区的数量。

ULB-Miner[46](2018):
1.列表名字：utility list buffer(效用列表缓冲区);
2.列表结构：tid：表示包含X的事务T；
iutil：X在T中的效用；
rutil：X在T中的剩余效用；
SUL(X)：指示有关项集X的信息存储在效用列表缓冲区中的位置；
3.数据集：Connect, Chainstore, Chess, Foodmart, Kosarak, Retail；
4.优点和缺点：效用列表的数量减少了，内存消耗很小，并且在密集和稀疏数据集上的性能都很好；构造效用列表缓冲区的过程更加复杂。

LHUI-Miner[47](2019):
1.列表名字：local utility table (LU-list 局部效用列表);
2.列表结构：iutilPeriods：存储缩写项集X的最大LHUI（局部高效用项集）期限；
utilPeriods：存储缩写项集X的PLHUI（有希望的LHUI期）周期；
3.数据集：Mushroom, Retail, Kosarak, E-commerce；
4.优点和缺点：发现传统HUIM无法找到的有用模式，从而减少运行时间和内存消耗；动态调整参数具有挑战性。

其他列表

FHN[53](2016):
1.列表名字：positive and negative utility list (PNU 正面和负面效用列表);
2.列表结构：tid：事务标识符；
putil：事务中项集X的正项集之和；
nutil：事务中项集X的负项集之和；
rputil：剩余项的正值；
3.数据集：Mushroom, Retail, Chess, Accidents, Psumb, BMS-POS, T10I4D100K；
4.优点和缺点：不会生成任何候选项，也不需要执行多个耗时的数据库扫描；在稀疏数据集上的性能较弱。

MUHUI[75](2017):
1.列表名字：probability-utility list (PU-list 概率效用列表);
2.列表结构：tid：事务标识符；
prob：事务Tq中项集X的概率；
iu：事务Tq中项集X的效用；
ru：事务Tq中的项集X的剩余效用；
3.数据集：Foodmart, Accident, Retail, T10I4D100K；
4.优点和缺点：在效率和可伸缩性方面有明显的优势；在密集数据库Retail上的性能不佳。

HUOPM[49](2017):
1.列表名字：utility occupation list (UO-list 效用占用列表);
2.列表结构：tid：事务标识符；
uo：事务Tq中项集X的效用占用；
ruo：事务Tq中项集X的剩余效用占用；
3.数据集：BMSPOS2, Retail, Chess, Mushroom, T10I4D100K, T40I10D100K；
4.优点和缺点：在某种程度上，它为效用挖掘提供了新的研究视角； 没有过滤掉没有希望的项。

DMHUPS[50](2019):
1.列表名字：IUData-List;
2.列表结构：item：此节点包含的项的信息；
utility：事务中项的效用值之和；
extUB：作为长度为1的项集的项的扩展上限；
tidList：根据项的顺序包含事务ID和项位置的列表；
3.数据集：Chainstore, Kosarak, Retail, Accidents, Mushroom, Chess；
4.优点和缺点：在稀疏和密集数据集上表现良好；在某些情况下，内存很昂贵。

IMHUP[49](2017):
1.列表名字：index utility list (IU-list 索引效用列表);
2.列表结构：iitem：修订数据库中的第一项；
iutil：修订后的数据库中项集Xk的效用值；
index：索引号；
3.数据集：Accidents, Chain-store, Chess, Connect, Retail, T10I4D100K, T10I4D200K；
4.优点和缺点：减少效用列表之间的连接操作，而不生成任何候选对象；效用的上限不够严格。

图 2　基于列表(list)的 HUPM 方法

Fig. 2　List-based HUPM method

表 7　基于数据格式的 HUPM 方法

Tab. 7　Data format-based HUPM algorithm

| 算法名字 | 数据格式 | 数据集 | 优点 | 缺点 |
|---|---|---|---|---|
| UtilityLevel[55] (2010) | 水平表示 | D100K.C8.T6.S6.I5.N10K, D200K.C10.T8.S8.I7.N10, BMS-WebView-1, BMS-WebView-2 | 更直接，更简单。 | 生成大量候选序列并需要多次数据库扫描。 |
| SPHUTTP[56] (2017) | 水平表示 | Retail, Chess, Mushroom, T10I4D100K | 短期约束和效用指标可以大大减少找到的模式数量。 | 需要多次数据库扫描，浪费时间。 |
| FHM[10] (2015) | 垂直表示 | Chain-store, BMS, Kosarak, Retail | 不仅具有 HUI-Miner 的优点，而且减少了效用列表之间的连接操作。 | 比 HUI-Miner 稍微多消耗一些内存，在密集数据集上性能很差。 |
| CHUM[21] (2016) | 垂直表示 | Mushroom, Retail, Foodmart, Chain-store, T15I5D100K, T10I4D100K | 在密集，稀疏和真实数据集都有较好的性能。 | 占用内存较大。 |
| MHUI[14] (2018) | 垂直表示 | Chain, Kosarak, Pumsb, Accidents、Connect, Chess, T10I4D100K, T40I10D100K | 在中等长度和密集的基准数据集上有良好的性能。 | 对最小效用阈值的选择不敏感。 |
| HUI-MMUTID[61] (2015) | 索引 | Retail, T10I4D100K | 它可以有效地识别数据库中的所有 HUIs，并避免“稀有项问题”。 | 内存消耗较大。 |
| ISR-MOEA[58] (2019) | 索引 | Chess, Mushroom, Connect, Accidents, Powerc, OnlineRetai, D300kN3k, D200kN3k, D300kN3k | 进一步提高用户满意度。 | 所使用的随机初始化根本不能被覆盖。 |

## 3.1　增量方法

近年来，在各种应用领域中产生了越来越多的数据，随着时间的推移，数据的特征和数量不断变化。因为静态方法必须在每次输入新数据时从头开始自己的挖掘操作，所以它们会遭受非常大的计算开销。因此，为了更好地处理增量数据，研究人员提出了基于增量数据的 HUPM 方法，该方法仅处理新输入数据而无须额外的数据库扫描，并将其反映到先前的处理中而没有任何错误。

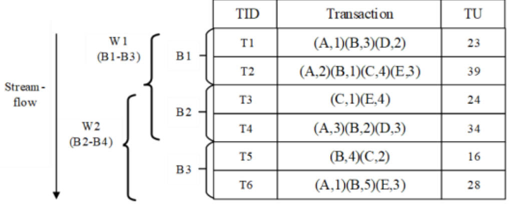
FUP-HU[64]算法是一种基于 Apriori 的增量 HUPM 算法。当新事务插入到原始数据库中时，该算法根据它们是原始数据库中还是新事务中的高事务加权效用项集将项集分为四个部分。然后处理每个部分以自己的方式找到的有效项集。PRE-HUI[65]算法是 FUP-HU 算法的变体，它使用两种类型的阈值，即上限阈值和下限阈值，并使用新概念来预测在增量数据上模式状态的变化，称为 Pre-large concept。此算法保证了比 FUP-HU 算法更好的性能，但它不是一种准确的方法，由于设置了较低阈值而丢失了一定的模式。

因为基于 Apriori 的算法会产生大量候选项，运行时间长，内存消耗大。研究人员开发了基于树和基于列表的增量 HUPM 算法。IHUP[3]算法在单个数据库扫描中构建自己的树结构，并提取高效用模式的候选者。然后，算法通过额外的数据库扫描识别候选者的实际高效用模式。每当输入新的增量数据时，该方法处理它们，更新先前构造的树结构，并执行其自己的挖掘操作。IMHAUI[66]算法提出了增量高效用平均项集树(incremental high average utility itemset tree, IHAUI-Tree)的新树结构，以维护增量数据库的信息，以便可以在没有多个数据库扫描的情况下挖掘 HAUIs。该算法使用路径调整方法作为重建技术之一，以保持 IHAUI-Tree 的紧凑性。LIHUP[68]算法通过单次扫描构建全局数据结构，根据最佳排序顺序重建数据结构，并在重组步骤中更新效用信息，以有效地从增量数据库中挖掘 HUIs。IIHUM[67]算法基于索引列表从增量数据库中挖掘 HUIs 而没有任何候选生成，并设计重组和修剪技术以更有效地处理增量数据。TOPK-HUP-INS[80]算法通过效用列表存储事务数据信息，并对添加新事务后的数据库进行挖掘，得到前 $k$ 个高效用模式。该算法主要由三个部分组成：a)创建“1-模式”效用列表；b)挖掘效用值前 $k$ 个高效用模式；c)对新事务的挖掘处理。

### 3.2　基于滑动窗口模型的方法

在滑动窗口模型中，数据流被分成批次。这些批次包含许多事务。该模式使用包含有限数量批次的窗口来维护数据结构中的最新批次。窗口大小由用户指定。输入新数据时，旧批次中的大多数先前数据将从窗口中排除，新数据将作为最新批次输入窗口。接下来，解决在数据流上的滑动窗口中挖掘 HUPM 的问题。假设数据流 $DS=\{T_1,T_2,\cdots,T_n\}$ 是一组事务，$I=\{i_1,i_2,\cdots,i_m\}$ 是一组项，而模式 $P$ 是由一组 $I$ 中的项组成，其中 $P\subseteq I, 1\le k\le m$。$P$ 的长度表示为 $k$，并且 $k$ 项的集合表示项集是长度为 $k$ 的模式。数据流 $DS$ 中的每个事务 $T_i$(其中 $1\le i\le n$)表示第 $i$ 个到达的事务。在滑动窗口模型中，每个窗口 $W_k$ 由固定数量的相等大小和非重叠批次组成。

图 3 是分成四个批次{$B_1$, $B_2$, $B_3$}的数据流的示例，其中每个滑动窗口由三个批次组成。在该示例中，存在两个滑动窗口，$W_1$ = {$B_1$, $B_2$}和 $W_2$ = {$B_2$, $B_3$}。这里，$W_1$ 是初始滑动窗口。此外，$W_2$ 是当第一个窗口 $W_1$ 已满，并且通过删除最旧的批次并插入新批次滑动的结果。也就是说，在滑动窗口模型中，算法在当前窗口中仅使用固定数量的最近批次。




| TID | Transaction | TU |
|---|---|---|
| T1 | (A,1)(B,3)(D,2) | 23 |
| T2 | (A,2)(B,1)(C,4)(E,3) | 39 |
| T3 | (C,1)(E,4) | 24 |
| T4 | (A,3)(B,2)(D,3) | 34 |
| T5 | (B,4)(C,2) | 16 |
| T6 | (A,1)(B,5)(E,3) | 28 |

图 3　数据流示例

Fig. 3　Data flow example

因此，基于此模型的挖掘算法始终可以在窗口中保留最新信息。THUIMine[69]算法是该领域中第一个在资源受限环境中对数据流利用 HUPM 的算法，但它在运行时和内存使用方面存在许多缺点，因为它与 Apriori 算法类似。在此基础上，提出了基于树结构的 HUMPS[70]算法来克服 THUI-Mine 算法的缺点。GUIDEsw[71]算法中的窗口是一个时间敏感窗口，用于修复事务的大小。挖掘该模型的算法不仅具有较高的实用价值，而且最大化概念数据流，进一步提高了准确性和运行时间。SHUGrowth[72]算法使用树结构 SHU-Tree，它在全局树中有一个节点效用计数器。计数器的每个效用值与当前窗口中的每个批次相关联，即，如果当前窗口中有 $n$ 个批次，则节点效用在计数器中为 $n$。

基于 Top-k 模式，Vert-top-k-DS[73]算法提出了一种新的数据结构 iList，可以快速插入和删除窗口。T-HUDS[74]算法使用 HUDS-Tree 的紧凑数据结构，用于在滑动窗口中动态维护事务的压缩版本。基于此结构，该算法在不指定 *minutil* 的情况下查找数据流中的 Top-k HUIs。

基于顺序模式，HUSP-Stream[75]算法使用垂直表示项效用列表(item utility lists, ItemUtilLists)和外部效用树来模拟当前窗口中 shell 的基本信息，逐步表示数据流中的完整 HUSP。HUSP-UT[76]算法改进了上述算法并使用了新的数据结构 UT-Tree。该算法优于最先进的 shell 流算法。HUPMS[77]算法基于高效的流树，通过将数据流的重要信息捕获到 shell 树中，使用模式增长方法挖掘当前窗口中的所有 HUIs 对于数据流上的增量和交互式挖掘非常有效。

### 3.3　基于时间衰减模型的方法

时间衰减模型旨在通过区分最近事务与旧事务的重要性来从数据流中找到最新的相关信息。该模型使用用户指定的衰减系数 $f$ 来降低事务随时间的重要性。使用此模型，如果旧事务项集的频率很高，则可以在最近的挖掘过程中考虑它。因此，该模型比滑动窗口模型更可靠和有效，因为旧事务的信息并未完全从挖掘过程中排除。该方法可以应用于 HUPM，并且根据交易的到达时间来减少交易的效用以挖掘最新的 HUIs。

基于该模型的频繁项集挖掘方法通过将它们乘以衰减因子来减少旧事务中项集的出现次数，即频率。设 $S=\{T_1,T_2,\cdots,T_i\}$ 是由多个事务组成的数据流，$X$ 是从 $S$ 中生成的一组项。当每个事务中 $X$ 的频率 $T_k$ 表示为 $\mathrm{freq}(T_k,X)$，基于时间衰减模型，$S$ 中 $X$ 的衰减频率表示为 $\mathrm{dfreq}(X)$，计算如下：

$$\mathrm{dfreq}(X)=\sum_{k=1}^{n}\mathrm{freq}(T_k,X)\times f^{n-k} \tag{9}$$

GENHUI[78]算法基于时间衰减模型，基于事务的到达时间来减少事务的效用，以便为最新数据分配比旧数据更多的权重。同时，定期更新其树数据结构，以将最新的效用信息反映到树结构中。每当执行更新过程时，算法都会从树中修剪过时的节点，以便仅保留对挖掘结果具有高影响的信息。HAUPM[79]算法通过考虑给定数据的时间因子来发现重要的最近模式信息，挖掘最近的 HUIs。为了促进有效的挖掘过程，该算法设计并使用新的数据结构，阻尼平均树和事务效用列表挖掘 HAUIs。

### 3.4　基于地标模型的方法

在一些应用中，用户可能希望从过去的时间点(地标)平等地处理所有数据并从数据流中发现长期模式，地标模型用于此目的。地标窗口模型的数据会随着时间的推进不断增多，存储了较多旧数据，在某些情况会影响挖掘结果，并且会增大内存的消耗[85]。

MAHUSP[35]算法基于具有里程碑意义的窗口挖掘 HUIs。它是一种基于数据流增量挖掘 shell 的新方法，它不仅可以识别最近的 shell，而且可以长时间识别 shell。该算法使用新颖且紧凑的数据结构 MAS-Tree，用于在数据流上存储潜在的 shell。同时，使用两种有效的内存自用机制来解决现有内存

不足以向 MAS-Tree 添加新的潜在 shell 的问题。因此，算法 MAHUSP 可以有效地在数据流上找到 shell，具有很高的召回率和准确性。

## 4　实验指标

算法的性能必须在实验上进行验证，这一章主要描述了关于 HUPM 的实验指标，包括常用的实验方法、实验数据集和实验的评价指标。

### 4.1　实验方法

在 HUPM 中，主要的实验方法从两个方面阐述，分别为静态方法、动态方法。动态方法包括增量方法和数据流方法等。

a)静态数据是基本保持稳定的数据，它们在很长的一段时间内不会变化，一般不随运行而变。数据库大小的快速扩展已导致有效数据挖掘技术的多种方法的发展。高效用项集挖掘用于从包含事务数据的大型数据库[1,4,9,10,12]中查找项集(即项集或模式)，其中涉及它们之间的特定关系。

b)在增量数据库中，增加、删除和修改是非常频繁的操作，在这种情况下，批处理过程必须重新扫描整个更新的数据库以维护最新信息，例如，客户 $X$ 购买了四支铅笔和一块橡皮，而客户 $Y$ 购买了一个电脑鼠标。一段时间后，客户 $Z$ 可能会来买两个面包和一瓶牛奶，客户 $X$ 可能会来还两支铅笔，客户 $Y$ 可能会来退还鼠标。因此，增量数据库[3,6,7,19,24]中高效用挖掘可以根据频繁添加的新事务和修改或删除旧事务[3]来挖掘项集。

数据流由以实时方式顺序到达的连续有序数据组成。数据流挖掘有许多种应用程序，例如在线电子商务或事务流中的知识发现，网络流分析，传感器数据监视以及 Web 日志和点击流挖掘。与传统的数据库不同，数据流具有一些特殊的属性：连续，无界，具有高速且随时间变化的数据分布。在数据流[35,36,69,71,72]上高效用项集挖掘可以避免产生太多的模式，并且可以过滤掉没有希望的模式。

### 4.2　典型数据集

在 HUPM 中，研究者经常使用的典型数据集有稀疏数据集、密集数据集(当数据库的密度小于 1%时，数据集的性质就会稀疏[50]。)和合成数据集。具体数据集特征如表 8 所示。

表 8　数据集特征

Tab. 8　Data set features

| 数据集 | 事务数量 | 不同项的数量 | 平均事务长度 | 最大事务长度 | 密集度(%) | 类型 | 应用 |
|---|---|---|---|---|---|---|---|
| Accidents | 340,183 | 468 | 33.8 | 51 | 7.2239 | Dense | MHUI[14], FOSHU[16], MAU-Growth[33], MUHUI[52], IMHUP[54] |
| BMS | 59,601 | 497 | 2.51 | | 0.5052 | Sparse | UP-growth[27], CRUSP[51] |
| Chain-store | 1,112,949 | 46,086 | 7.26 | 170 | 0.0156 | Sparse | CHUIMiner[11], MHUI[14], CHUM[21], MAU-Growth[33], HUP-Miner[44] |
| Chess | 3,196 | 75 | 37 | 37 | 49.3333 | Dense | CHUIMiner[11], MHUI[14], FOSHU[16], CHUM[21], UP-growth[27] |
| Connect | 67,557 | 129 | 43 | 43 | 33.3333 | Dense | CHUIMiner[11], MHUI[14], ULB-Miner[46], IMHUP[54] |
| Foodmart | 4,141 | 1,559 | 4.4 | 14 | 0.2822 | Sparse | PRE-HUI-DEL[5], CHUIMiner [11], CHUM[21], HIMU[31], ULB-Miner[46] |
| Kosarak | 990,000 | 41,270 | 8.09 | 2,498 | 0.0196 | Sparse | MHUI[14], HUP-Miner[44], ULB-Miner[46], LHUI[47] |
| Mushroom | 8,124 | 119 | 23 | 23 | 19.3277 | Dense | CHUIMiner[11], FOSHU[16], CHUM[21], HIMU[31], dHAUIM[32] |
| Pumsb | 49,046 | 2,113 | 74 | | 3.5021 | Dense | MHUI[14], FOSHU[16] |
| Retail | 88,162 | 16,47 | 10,30 | 76 | 0.0625 | Sparse | CHUIMiner [11], FOSHU[16], CHUM[21], MAU-Growth[33], HUP-Miner[44] |
| T10I4D100K | 100,000 | 870 | 10.1 | 29 | 1.1609 | Dense | MHUI[14], CHUM[21], HUP-Miner[44], HUOPM[49], MUHUI[52] |
| T40I10D100K | 100,100 | 942 | 39.6 | 77 | 4.2038 | Dense | MHUI[14], HUP-Miner[44], HUOPM[49] |

a) 稀疏数据集。Kosarak 数据集包含匈牙利在线新闻门户网站的网络点击流数据。Chain-store 数据集是从主要市场生成的大型零售数据集，已经包含单位利润信息和购买数量。Retail 数据集包含来自匿名比利时零售商店的零售市场购物篮数据，但是其大小比 Chain-store 小得多。对于其他数据集，通过使用对数正态分布生成项的外部效用，并在设置之间随机生成项的数量。

b) 密集数据集。BMS 数据集已经包含内部和外部效用。Connect 数据集包括联结的数量。Pumsb 是包含人口普查数据的密集数据集。Accidents 数据集包含匿名交通事故数据。Mushroom 数据集包含各种蘑菇种类的特征信息。

c) 合成数据集。数据集 T10I4D100K 到 T10I4D1000K 的数据集由 IBM Quest 合成数据生成代码生成，项数是恒定的，但事务的数目却逐渐增加。在从 T10N10000L1000 到 T40N40000L4000 的另一个数据集中，项数和平均事务时间逐渐增加。

### 4.3　评价指标

研究者评价 HUPM 经常用到的指标有运行时间、内存消耗、候选项集数量、可伸缩性、查全率、查准率等。实验中经常使用以下性能指标：

a) 运行时间[17,18,39,42,48]由输入时间，CPU 时间和输出时间等组成，在每个数据集上运行算法，同时减小 *minutil* 阈值，直到算法变得执行时间太长，内存不足或观察到明显的赢家为止，来判断算法的性能。

b) 内存使用情况[18,28,39,41,42]。对于数据集，观察内存的使用情况，确定使用成本的高低。剪枝策略、效用列表的数量等因素都会影响内存的使用。

c) 候选项集的数量[53,56]。通过不同算法在不同数据集上生成的 HTWUIs 和 HUIs 的数量进行评估，以进一步理解所比较算法的性能。算法中不同的修剪策略会直接影响候选项集数量的多少。

d) 可伸缩性分析[15,17,39,40,41]。在合成数据集上比较算法的可伸缩性，随着事务数量的变化，比较算法的运行时间、内存使用等指标，即所提出的算法在不同的数据集大小和参数下是否具有良好的可扩展性。

e) 时间复杂度[48,68]。在 MHAI[48]中，令 $n_t$ 为给定数据库中的事务数，$n_i$ 为每个事务中的项数。构造初始 HAI-Lists 的时间复杂度为 $O(2\times n_t\times n_i + n_t\times n_i \log 2\ n_i)$。通过递归构造 HAI-List 来挖掘所有项集需要 $O(n_t\times(2^{n_i}\text{-}1\text{-}n_i))$的时间复杂度。在 LIHUP[68]算法中，令 $N_o$ 和 $N_i$ 为分别由 $N_m$ 个项组成的原始数据库和增加的数据库中的事务数。将所有事务插入原始或更新的数据库中的过程的时间复杂度为 $O(N_o\times N_m)$或 $O(N_i\times N_m)$。由于候选项集需要额外的数据库扫描以从 $N_c$ 个候选项中识别出实际的高效用模式，因此用于增加的数据库的时间复杂度对于原始数据库为 $O(N_o\times N_m\times N_c)$或 $O((N_o + N_i))\times N_m\times N_c$。

## 5　下一步研究方向

高效用模式挖掘问题的解决方法虽然得到了一定的发展，

但是现有的方法仍然存在着不足之处，这为研究者提供了下一步的研究方向：

a) 效用模式中存储着大量有效的信息，但是仍存在着大量冗余模式。针对减少冗余模式的紧凑高效用模式进行研究与分析，本文作者项目组下一步研究 Top-k 闭高效用模式的一阶段算法。该算法使用改进的 uList 结构，计算模式的真实和剩余效用来剪枝遍历空间，对结果集存储的 Top-k 缓存区内容实时更新，进行闭高效用模式生成。

b) 高效用模式由于生成大量的候选项集，而消耗了非常多的内存和时间。针对减少这两方面来分析，本文作者项目组下一步研究基于改进效用列表的增量算法。该算法应用了 ULB-Miner[46]算法中的效用列表缓冲区结构能大幅度地减少生成候选项集占用的内存。此算法从增量角度挖掘项集，更加的贴合实际应用。

c) 在高效用模式挖掘中，可伸缩性是重要研究问题。可伸缩性就是在一定的维度下，处理更多的数据。可采取以下措施来解决 HUPM 中有关可伸缩性的问题：(1)并置：通过并置数据和代码，减少因获取所需数据而产生的必要开销；(2)缓存：如果数据和代码不能并置，就缓存数据，以减少与其使用相关的开销；(3)通过分割处理代码、并置相关的分区，尽可能将相关的处理过程集中在一起，可以减少单个工作单元的处理时间。

d) 高效用效用模式挖掘现阶段主要应用于单个数据流，没有多个数据流的情况。可以通过并行的方法同时处理多个数据流。首先需要对程序进行并行化处理，也就是说将工作各部分分配到不同处理进程(线程)中。从理论上讲，在 $n$ 个并行处理的执行速度可能会是在单一处理机上执行的速度的 $n$ 倍。

e) 随着大数据、云计算等技术广泛应用于实际生活中，在 HUPM 中，静态数据挖掘已不能满足人们的需求。在大数据环境下的 HUPM 中，可以使用 Spark 工具来存储数据，并实现了由单机到分布式的改造；可以使用 Storm 工具实时地处理大数据流；还可以使用 Python 中的 Scrapy 框架来爬取网站数据，提取结构性数据，并且这是一个非常好的从互联网上抓取数据的 Web 框架。利用这些工具、框架可以解决高效用模式挖掘中的计算、存储资源、容错、优化负载均衡等问题。

## 6 结束语

高效用模式挖掘(HUPM)是效用挖掘中一项至关重要的任务。到目前为止，已经为 HUIM 的任务广泛提出了许多技术和方法。本文介绍了 HUPM 的基本概念、实例、相关概念，对高效用关键技术进行了阐述，分别从基于 Apriori、基于树、基于映射、基于数据格式、基于索引、基于列表的方法论述。分析了算法的工作流程、用途、使用的数据集以及优缺点。在增量数据库和数据流上，分析描述了 HUPM 的方法。最后提出下一步的研究方向，本文提到的大多数算法还是应用于静态数据中，使之运用于数据流是以后的工作之重。

## 参考文献：


[1] Liu Y, Liao W. Choudhary A. A two-phase algorithm for fast discovery of high utility itemsets [J]. Advances in Knowledge Discovery and Data Mining, 2005, 3518: 689–695.

[2] Liu Y, Liao W, Choudhary A. A fast high utility itemsets mining algorithm [C]// The 1st International Workshop on Utility-Based Data Mining, 2005: 90-99.

[3] Ahmed C F, Tanbeer S K, Jeong B S, *et al.* Efficient tree structures for high utility pattern mining in incremental databases [J]. IEEE Transactions on Knowledge and Data Engineering, 2009, 21 (12): 1708–1721.

[4] Lin C W, Hong T P, Lu W H. An effective tree structure for mining high utility itemsets [J]. Expert Systems with Applications, 2011, 38 (6): 7419–7424.

[5] Lin C W, Lan G C, Hong T P. Efficient updating of discovered high-utility itemsets for transaction deletion in dynamic databases [J]. Intelligent Data Analysis, 2015, 19 (1): 43–55.

[6] Lin C W, Hong T P, Lan G C, *et al.* Incrementally mining high utility patterns based on pre-large concept [J]. Applied Intelligence, 2014, 40 (2): 343–357.

[7] Lee J, Yun U, Lee G, *et al.* Efficient incremental high utility pattern mining based on pre-large concept [J]. Engineering Applications of Artificial Intelligence, 2018, 72: 111-123.

[8] Hong T P, Hsu J H, Lan G C, *et al.* High Utility Partial Periodic Pattern Mining [C]// ACM International Conference Proceeding Series, 2017.

[9] Liu M, Qu J. Mining high utility itemsets without candidate generation [C]// The ACM International Conference on Information and Knowledge Management, 2012: 55–64.

[10] Fournier-Viger P, Wu C W, *et al.* FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning [C]// The International Symposium on Methodologies for Intelligent Systems, 2014: 83–92.

[11] Wu C W, Fournier-Viger P, Gu J Y, *et al.* Mining closed+high utility itemsets without candidate generation [C]// 2015 Conference on Technologies and Applications of Artificial Intelligence (TAAI 2015) . IEEE, 2015: 187–194.

[12] Zida S, Fournier-Viger P, Lin J C W, *et al.* EFIM: a highly efficient algorithm for high-utility itemset mining [C]// Mexican International Conference on Artificial Intelligence. Springer International Publishing, 2015: 530–546.

[13] Peng A Y, Koh Y S, Riddle P. mHUIMiner: A fast high utility itemset mining algorithm for sparse datasets [C]// PacificAsia Conference on Knowledge Discovery and Data Mining. Springer, Cham, 2017: 196–207.

[14] Krishnamoorthy S. Efficient mining of high utility itemsets with multiple minimum utility thresholds [J]. Engineering Applications of Artificial Intelligence, 2018, 69: 112-126.

[15] Duong Q H, Liao B, Fournier-Viger P, *et al.* An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies [J]. Knowledge-Based Systems, 2016, 104: 106-122.

[16] Fournier-Viger P, Zida S. FOSHU: Faster On-Shelf High Utility Itemset Mining with or without Negative Unit Profit [C]// The 30th Annual ACM Symposium. ACM. 2015: 857-864.

[17] Gan W S, Lin J C W, Fournier-Viger P. Mining High-Utility Itemsets with Both Positive and Negative Unit Profits from Uncertain Databases [C]// Lecture Notes in Computer Science, 2017: 434-446.

[18] Lin J C W, Gan W, Hong T P, *et al.* Efficient algorithms for mining up-to-date high-utility patterns [J]. Advanced Engineering Informatics, 2015, 29 (3): 648–661.

[19] Lin J C W, Gan W, Hong T P, *et al.* An incremental high-utility mining algorithm with transaction insertion [J]. The Scientific World Journal, 2015: 1-15.

[20] Lan G C, Hong T P, Tseng V S, *et al.* Applying the maximum utility measure in high utility sequential pattern mining [J]. Expert Systems with Applications, 2014, 41 (11): 5071–5081.

[21] Sahoo J, Das A K, Goswami A. An efficient fast algorithm for discovering closed+high utility itemsets [J]. Applied Intelligence, 2016, 45 (1): 44–74.

[22] Gan W S, Lin J C. W, Fournier-Viger P, *et al.* A Survey of Utility Oriented

Pattern Mining [J]. JOURNAL OF LATEX CLASS FILES, 2018, 6 (1) .
[23] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases [J]. In Acm sigmod record, 1993, 22 (2): 207–216.
[24] Hong T P, Lee C H, Wang S L. An Incremental Mining Algorithm for High Average-Utility Itemsets [C]// International Symposium on Pervasive Systems. IEEE, 2010: 421-425.
[25] Lin J C W, Gan W S, Fournier-Viger P, *et al.* Efficient algorithms for mining high-utility itemsets in uncertain databases [J]. Knowledge-Based Systems, 2016, 96: 171-187.
[26] Wu C W, Fournier-Viger P, Yu P S, *et al.* Efficient mining of a concise and lossless representation of high utility itemsets [C]// International Conference on Data Mining. IEEE, 2011: 824-833.
[27] Tseng V S, Wu C W, *et al.* UP-growth: An efficient algorithm for high utility itemset mining [C]// The ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2010: 253–262.
[28] Tseng V S, Wu C W, *et al.* Efficient algorithms for mining high utility itemsets from transactional databases [J]. IEEE Transactions on Knowledge and Data Engineering, 2013, 25 (8): 1772–1786.
[29] Erwin A, Gopalan R P, Achuthan N. Efficient mining of high utility itemsets from large datasets [C]// Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Berlin, Heidelberg, 2008: 554-561.
[30] Yin J, Zheng Z, Cao L. USpan: an efficient algorithm for mining high utility sequential patterns [C]// The 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012: 660-668.
[31] Gan W S, Lin J C W, Fournier-Viger P, *et al.* More Efficient Algorithms for Mining High-Utility Itemsets with Multiple Minimum Utility Thresholds [J]. Lecture Notes in Computer Science, 2016, 9827: 71-87.
[32] Truong T, Duong H, Le B, *et al.* Efficient Vertical Mining of High Average-Utility Itemsets Based on Novel Upper-Bounds [J]. IEEE Transactions on Knowledge and Data Engineering, 2019, 31 (2): 301-314.
[33] Kim D, Yun U. Efficient mining of high utility pattern with considering of rarity and length [J]. Applied Intelligence, 2016, 45 (1): 152-173.
[34] Heungmo R, Unil Y. Top-k high utility pattern mining with effective threshold raising strategies [J]. Knowledge-Based Systems, 2015, 76: 109–126.
[35] Zihayat M, Chen Y, An A. J. Memory-adaptive high utility sequential pattern mining over data streams [J]. Mach Learn, 2017, 106: 799–836.
[36] Lin F, Le W, Jin B. UT-Tree: Efficient mining of high utility itemsets from data streams [J]. Intelligent Data Analysis, 2013, 17: 585–602.
[37] Pei J, Han J, Mortazavi-Asl B, *et al.* PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth [C]// The 17th international conference on data engineering. IEEE Computer Society, 2001: 215–224.
[38] Nguyen L T, Nguyen P, Nguyen T D, *et al.* Mining high-utility itemsets in dynamic profit databases [J]. Knowledge-Based Systems, 2019, 175: 130-144.
[39] Singh K, Singh S S, Kumar A, *et al.* CHN: an efficient algorithm for mining closed high utility itemsets with negative utility [J]. IEEE Transactions on Knowledge and Data Engineering, 2018.
[40] Dam T L, Li K, Fournier-Viger P, *et al.* CLS-Miner: efficient and effective closed high utility itemset mining [J]. Frontiers of Computer Science, 2018: 1-25.
[41] Singh K, Kumar A, Singh S S, *et al.* EHNL: An efficient algorithm for mining high utility itemsets with negative utility value and length constraints [J]. Information Sciences, 2019, 484: 44-70.
[42] Singh K, Shakya H K, Singh A, *et al.* Mining of high - utility itemsets with negative utility [J]. Expert Systems, 2018.
[43] Lin J C W, Gan W, Hong T P. A fast maintenance algorithm of the discovered high-utility itemsets with transaction deletion [J]. Intelligent Data Analysis, 2016, 20 (4): 891–913.
[44] Krishnamoorthy S. Pruning strategies for mining high utility itemsets [J]. Expert Systems with Applications, 2015, 42 (5): 2371– 2381.
[45] Zaki M J. Scalable algorithms for association mining [J]. IEEE Transactions on Knowledge and Data Engineering, 2000, 12: 372–390.
[46] Duong Q H, Fournier-Viger P, Ramampiaro H, *et al.* Efficient high utility itemset mining using buffered utility-lists [J]. Applied Intelligence, 2017.
[47] Fournier-Viger P, Zhang Y. M, Lin J. C. W, *et al.* Mining local and peak high utility itemsets [J]. Information Sciences, 2019, 481: 344–367.
[48] Yun U, Kim D. Mining of high average-utility itemsets using novel list structure and pruning strategy [J]. Future Generation Computer Systems, 2017, 68: 346-360.
[49] Gan W S, Lin J C W, Fournier-Viger P, *et al.* HUOPM: High-Utility Occupancy Pattern Mining [J]. IEEE Transactions on Cybernetics, 2018.
[50] Prasad J B, Jen-Wei H. DMHUPS: Discovering Multiple High Utility Patterns Simultaneously [J]. Knowledge and Information Systems, 2018.
[51] Buffett S. Candidate List Maintenance in High Utility Sequential Pattern Mining [C]// IEEE International Conference on Big Data, 2018.
[52] Lin J C W, Gan W S, Fournier-Viger P, *et al.* Efficiently mining uncertain high-utility itemsets [J]. Soft Computing, 2017, 21 (11): 2801-2820.
[53] Lin J C W, Gan W S, Fournier-Viger P. FHN: An efficient algorithm for mining high-utility itemsets with negative unit profits [J]. Knowledge-Based Systems, 2016, 111: 283-298.
[54] Ryang H, Yun U. Indexed list-based high utility pattern mining with utility upper-bound reduction and pattern combination techniques [J]. Knowledge and Information Systems, 2017, 51 (2): 627–659.
[55] Ahmed C F, Tanbeer S K, Jeong B S. A novel approach for mining high-utility sequential patterns in sequence databases [J]. ETRI journal, 2010, 32 (5): 676–686.
[56] Lin J C W, Zhang J X, Fournier-Viger P, *et al.* A two-phase approach to mine short-period high-utility itemsets in transactional databases [J]. Advanced Engineering Informatics, 2017, 33: 29-43.
[57] Zaki M J. Scalable algorithms for association mining [J]. IEEE Transactions on Knowledge and Data Engineering, 2000, 12 (3): 372–390.
[58] Zhang L, Yang S S, Wu X P, *et al.* An indexed set representation based multi-objective evolutionary approach for mining diversified top-k high utility patterns [J]. Engineering Applications of Artificial Intelligence, 2019, 77: 9-20.
[59] Song W, Zhang Z, Li J. A high utility itemset mining algorithm based on subsume index [J]. Knowledge and Information Systems, 2016, 49 (1): 315–340.
[60] Song W, Yang B, Xu Z. Index-BitTableFI: An improved algorithm for mining frequent itemsets [J]. Knowledge-Based Systems, 2008, 21 (6): 507–513.
[61] Lin J C W, Gan W S, Fournier-Viger P, *et al.* Mining High-Utility Itemsets with Multiple Minimum Utility Thresholds [C]// ACM International Conference Proceeding Series, 2015: 9-17.
[62] Cheng J, Ke Y, and Ng W. A survey on algorithms for mining frequent itemsets over data streams [J]. Knowledge and Information Systems, 2008, 16 (1): 1-27.
[63] Jiang N, Gruenwald L. Research Issues in Data Stream Association Rule Mining [J]. ACM SIGMOD Record, 2006, 35 (1): 14-19.
[64] Lin C W, Lan G C, Hong T P. An incremental mining algorithm for high utility itemsets [J]. Expert Systems with Applications, 2012, 39 (8):

7173-7180.

[65] Lin C W, Hong T P, Gan W, *et al.* Incrementally updating the discovered sequential patterns based on pre-large concept [J]. Intelligent Data Analysis, 2015, 19 (5): 1071-1089.

[66] Kim D, Yun U. Efficient algorithm for mining high average-utility itemsets in incremental transaction databases [J]. Applied Intelligence, 2017, 47 (1): 114-131.

[67] Yun U, Nam H, Lee G, *et al.* Efficient approach for incremental high utility pattern mining with indexed list structure [J]. Future Generation Computer Systems, 2019, 95: 221-239.

[68] Yun U, Ryang H, Lee G, *et al.* An efficient algorithm for mining high utility patterns from incremental databases with one database scan [J]. Knowledge-Based Systems, 2018, 124: 188-206.

[69] Chu C, Tseng V, Liang T. An efficient algorithm for mining temporal high utility itemsets from data streams [J]. Journal of Systems and Software, 2008, 81 (7): 1105-1117.

[70] Chen L, Mei Q. Mining frequent items in data stream using time fading model [J]. Information Sciences (ISCI) , 2014, 257: 54-69.

[71] Shie B E, Yu P S, Tseng V S. Efficient Algorithms for Mining Maximal High Utility Itemsets from Data Streams with Different Models [J]. Expert Systems with Applications, 2012, 39 (17): 12947-12960.

[72] Ryang H, Yun U. High utility pattern mining over data streams with sliding window technique [J]. Expert Systems With Applications, 2016, 57: 214-231.

[73] Dawar S, Sharma V, Goyal V. Mining top-k high-utility itemsets from a data stream under sliding window model [J]. Applied Intelligence, 2017, 47 (4): 1240-1255.

[74] Zihayat M, An A J. Mining top-k high utility patterns over data streams [J]. Information Sciences, 2014, 285: 138-161.

[75] Zihayat M, Wu C W, An A J, *et al.* Mining High Utility Sequential Patterns from Evolving Data Streams [C]// ASE BigData and SocialInformatics. ACM, 2015.

[76] Tang H, Liu Y, Wang L. A New Algorithm of Mining High Utility Sequential Pattern in Streaming Data [J]. International Journal of Computational Intelligence Systems, 2019, 12 (1): 342-350.

[77] Farhan A C, Khairuzzaman T S, Byeong-Soo J, *et al.* Interactive mining of high utility patterns over data streams [J]. Expert Systems with Applications, 2012, 39 (15): 11979-11991.

[78] Kim D, Yun U. Mining high utility itemsets based on the time decaying model [J]. Intelligent Data Analysis, 2016, 5 (20): 1157-1180.

[79] Yun U, Kim D, Yoon E, *et al.* Damped window based high average utility pattern mining over data streams [J]. Knowledge-Based Systems, 2018, 144: 188-205.

[80] 吴 倩，王林平，罗相洲等. 动态数据库中增量 Top-k 高效用模式挖掘算法 [J]. 计算机应用研究, 2017, 34 (5): 1401-1405. (Wu Qian, Wang Linping, Luo Xiangzhou, *et al.* Incremental Top-k high utility pattern mining algorithm in dynamic database [J]. Application Research of Computers, 2017, 34 (5): 1401-1405.)

[81] 张全贵，曹阳，李志强. 一种频率约束的高效用模式挖掘算法 [J]. 计算机应用与软件, 2018, 35 (11): 266-271. (Zhang Quangui, Cao Yang, Li Zhiqiang. A High Utility Pattern Mining Algorithm with Frequency Constraints [J]. Computer Applications and Software, 2018, 35 (11): 266-271.)

[82] 赵林柳，吕鑫，陶飞飞. 基于 Top-k 的高效用模式挖掘算法 [J]. 计算机工程, 2019, 45 (5): 169-174, 181. (Zhao Linliu, Lv Xin, Tao Feifei. Efficient Algorithm for High Utility Pattern Mining Based on Top-k [J]. Computer Engineering, 2019, 45 (5): 169-174.)

[83] 吕存伟，黄德才，陆亿红. 含负项的高效用序列模式挖掘算法 [J]. 小型微型计算机系统, 2017, 38 (8): 1724-1729. (Lv Cunwei, Huang Decai, Lu Yihong. High Utility Sequential Pattern Mining with Negative Unit Profits [J]. Journal of Chinese Computer Systems, 2017, 38 (8): 1724-1729.)

[84] 吴倩，王林平，罗相洲等. 基于 MapReduce 的 top-k 高效用模式挖掘算法 [J]. 计算机应用研究, 2017, 34 (10): 2897-2900, 2932. (Wu Qian, Wang Linping, Luo Xiangzhou, *et al.* Top-k high utility pattern mining algorithm based on MapReduce [J]. Application Research of Computers, 2017, 34 (10): 2897-2900, 2932.)

[85] 王少峰，韩 萌，贾 涛等. 数据流高效用模式挖掘综述 [J]. 计算机应用研究, 2020, 37 (9) . (Wang Shaofeng, Han Meng, Jia Tao, *et al.* Survey of high utility pattern mining over data streams [J]. Application Research of Computers, 2020, 37 (9) .)